PROGRAMAREA FUNCȚIONALĂ. ELEMENTE FUNDAMENTALE

LAMBDA CALCULUS

1. λ expresii şi funcții

- o λ expresie poate fi un **nume**, o **funcție** sau **aplicarea unei funcții**.
  - **numele** – poate fi, în principiu, orice succesiune de caractere (diferite de spațiu).
  - **funcția** – este de forma:
    λnume.corp unde:
    - *corp* este o λ expresie
    - *nume* se numeşte variabilă legată a funcției

    Obs.: funcția nu are nume.
  - **aplicarea** unei funcții – este de forma:
    (expresie1 expresie2) unde:
    - expresie1 - reprezintă o funcție
    - expresie2 – reprezintă argumentul.

    Obs.: aplicarea reprezintă de fapt o concretizare a funcției.

Exemple:

- **funcția identitate:** λx.x
  Aplicații: (λx.x a) => a
  (λx.x λx.x) => λx.x
- **autoaplicația:** λa.(a a)
- Aplicații: (λa.(a a) λx.x) => (λx.x λx.x) => λx.x
  (λa.(a a) λa.(a a)) => (λa.(a a) λa.(a a)) => ….

2. β reducerea

- Asocierea numelui unei funcții:
  def identitate = λx.x
  def auto_ aplică = λa.(a a)

  Expresia: (nume argument) reprezintă aplicarea numelui respectiv la argumentul specificat.

- În λ calculus, înlocuirea variabilei legate cu argumentul specificat în aplicație, ori de câte ori apare în corpul funcției, se numește **β reducere**.

- Notații:
  (funcție argument) => expresie
  (funcție argument) => …. => expresie

- Exemple de funcții:

- selectarea primului argument dintre două argumente:
  def sel_prim = λprim.λsecund.prim

  Exemplu de aplicație:
  ((sel_prim arg1) arg2) ==
  ((λprim.λsecund.prim arg1) arg2) =>
  (λsecund.arg1 arg2) => arg1
  Obs.: aplicată unei perechi de argumente, funcția returnează întotdeauna primul argument; cel de-al doilea este practic ignorat.
  ✓ se poate renunța la paranteze:
  sel_prim arg1 arg2
  ✓ în λcalculus – funcții încuibate
  λprim.λsecund.prim -> λsecund.arg1 -> arg1

- selectarea celui de-al doilea argument din cele două argumente:

  def sel_secund = λprim.λsecund.secund

  Exemplu de aplicație:
  sel_secund arg1 arg2 ==
  λprim.λsecund.secund arg1 arg2 =>
  λsecund.secund arg2 => arg2

- construirea unui cuplu de valori:

  def constr_pereche = λprim.λsecund.λf.(f prim secund)

  ✓ Aplicăm această funcție succesiv unei perechi *arg1* şi *arg2*:
  constr_pereche arg1 arg2 ==
  λprim.λsecund.λf.(f prim secund) arg1 arg2 =>
  λsecund.λf.(f arg1 secund) arg2 =>
  λf.(f arg1 arg2)
  Funcția rezultată reprezintă cuplul construit.

  ✓ Aplicând funcția, mai departe, asupra lui *sel_prim* sau *sel_secund*, se poate selecta prima sau a doua valoare a cuplului:

  λf.(f arg1 arg2) sel_prim =>
  sel_prim arg1 arg2 =>…=> arg1
  respectiv:
  λf.(f arg1 arg2) sel_secund =>
  sel_secund arg1 arg2 =>…=> arg2

  ✓ Sintetizând:
  (constr_pereche arg1 arg2) sel_prim => … => arg1
  (constr_pereche arg1 arg2) sel_secund => … => arg2

3. **Legarea variabilelor. Variabile libere şi legate**

- Substituirea argumentelor în corpul funcțiilor se realizează fără probleme atunci când variabilele legate corespunzătoare funcțiilor dintr-o expresie sunt denumite în mod diferit.

  (λf.(f λx.x)) λa.(a a))

  Cele trei funcții implicate în expresia de mai sus au variabilele legate, respectiv f, x şi a.

  (λf.(f λx.x) λa.(a a)) =>

  (λa.(a a) λx.x) => (λx.x λx.x) => λx.x

  ✓ Expresia poate fi scrisă şi sub forma:

  (λf.(f λf.f)) λa.(a a))

  rezultatul substituirilor fiind λf.f

  ✓ La prima substituție, se înlocuieşte variabila legată f corespunzătoare funcției:

  λf. (f λf.f) cu λa.(a a)

  Aceasta presupune înlocuirea primului f din expresia (f λf.f); rezultă (λa.(a a) λf.f), care apoi se reduce în continuare ca mai sus.

- Domeniul unei variabile legate:

  Fiind dată funcția: λ nume.corp

domeniul variabilei legate *nume* se întinde asupra corpului funcției.

În expresia: (λf.λg.λa.(f (g a)) λg.(g g))

✓ domeniul variabilei legate *f* se întinde asupra expresiei:

λg.λa.(f (g a)) ;

✓ domeniul variabilei legate *g*, din stânga, se întinde asupra expresiei λa.(f (g a)) ;

✓ domeniul variabilei *g*, din dreapta, este expresia

(g g)

- Variabilă legată şi variabilă liberă într-o expresie:

Spunem că apariția unei variabile **v** într-o expresie **E** este **legată,** dacă ea apare într-o subexpresie a lui E, care este de forma **λv.E1** (adică, **v** apare în corpul unei funcții cu variabila legată numită **v**).

Exemple de expresii:

v (a b v) - v apare liber

λv.v(x y v) - v apare legat

v(λv.(y v) y) - v apare - liber (prima apariție)
- legat (ultima apariție)

- Concluzii:

✓ Fiind dată funcția

λnume.corp

domeniul variabilei legate *nume* se întinde asupra acelor secvențe ale corpului în care apariția variabilei *nume* este liberă.

✓ Fie funcția: λg.(g λh.(h (g λh.(h λg.(h g)))) g)

Pentru a stabili domeniul variabilei legate a funcției, *g*, analizăm corpul funcției

(g λh.(h (g λh.(h **λg.(h g)**))) g)

Aparițiile lui **g** în afara zonei marcate sunt libere.

- **Definirea riguroasă a β reducerii.**

Fiind dată o aplicație:

(λnume.corp argument)

se înlocuiesc toate aparițiile libere ale lui **nume** din **corp** cu **argument.**

Reluăm exemplul:

(λf.(f λf.f) λa.(a a))

Funcția aplicată este *λf.(f λf.f)* şi corpul ei este *(f λf.f)* => (λa.(a a) λf.f) => (λf.f λf.f) => λf.f

4. Conflicte de nume. α conversia

- Aplicând o β reducere, pot apare conflicte de nume.

Exemplu:

def f = λx.λy.(x y)

f y z == (λx.λy.(x y) y z) => (λy.(y y) z) => z z

Rezultatul este eronat.

Eroarea se poate corecta astfel:

(λx.λy1.(x y1) y z) => (λy1.(y y1) z) => y z

- Fiind dată o funcție

λnume1.corp

numele variabilei legate **nume1** precum şi toate aparițiile libere ale lui **nume1** din interiorul corpului funcției pot fi înlocuite cu un nou nume, **nume2**, cu condiția ca în λnume1.corp să nu apară nici o variabilă liberă numită **nume2**.

✓ Funcția λy.(x y) s-a transformat prin redenumire în funcția λy1.(x y1).

5. η reducerea

- η reducerea este o transformare care (ca şi β reducerea) permite înlocuirea unei λ expresii cu una echivalentă, mai simplă.
- fiind dată o funcție de forma:

  λ nume.(expresie nume)

  ea este echivalentă cu: *expresie*
- λ nume.(expresie nume) argument

  => (expresie argument)

6. Valori booleene şi expresii condiționale

- **λ calculus aplicat** – prevede atât valori logice şi numerice precum şi operațiile cu acestea.

✓ în C expresia condițională se scrie: cd? ex1: ex2

✓ modelarea valorilor logice în λ calculus: cu ajutorul funcțiilor *sel_prim*, *sel_secund* şi *constr_pereche*.

def cond = λe1.λe2.λc.(c e1 e2)

Aplicăm această funcție, succsesiv expresiilor ex1 şi ex2:

cond ex1 ex2 ==

λe1.λe2.λc.(c e1 e2) ex1 ex2 =>

λe2.λc.(c ex1 e2) ex2 => λc.(c ex1 ex2)

✓ valorile *true* şi *false* vor fi reprezentate prin funcțiile *sel_prim* şi respectiv *sel_secund* :

def true = λp.λs.p

def false = λp.λs.s

prin urmare :

cond ex1 ex2 true => … =>

λc.(c ex1 ex2) λp.λs.p =>

λp.λs.p ex1 ex2 => … => ex1

şi analog :

cond ex1 ex2 false => … =>

λc.(c ex1 ex2) λp.λs.s =>

λp.λs.s ex1 ex2 => … => ex2

- **operatorii logici NOT, AND şi OR** : se exprimă cu ajutorul expresiei conditionale :

- operatorul **NOT** :

def not = λx.(cond false true x)

Exemple:

not true ==λx.(cond false true x) true =>

cond false true true => … => false

respectiv

not false == λx.(cond false true x) false =>

cond false true false => … => true

- operatorul **AND** :

  def and = λx.λy.(cond y false x)

  Exemple:

✓ calculăm *true AND false*

(and true false) ==

λx.λy.(cond y false x) true false => … =>

cond false false true => … => false

✓ calculăm *false AND true*

(and false true) ==

λx.λy.(cond y false x) false true => … =>

cond true false false => … => false

✓ calculăm *NOT false AND true*

and (not false) true ==

λx.λy.(cond y false x) (λx.(cond false true x) false) true

=> … =>

λx.λy.(cond y false x) true true => … =>

cond true false true => … => true

- operatorul **OR** :

  def or = λx.λy.(cond true y x)

✓ exemplu: calculăm *true OR false*

(or true false) ==

λx.λy.(cond true y x) true false => … =>

cond true false true => … => true